

Side-Channel Analysis untuk Reverse Engineering

Studi Kasus Challenge CTF: Counter (Ekoparty), Wirth (Idsecconf), Why (Compfest)

{abrari, hrdn}@CySecIPB - 2016

Pendahuluan

Side-channel analysis atau side-channel attack merupakan teknik yang memanfaatkan informasi “side-channel” atau “informasi sampingan” dari suatu proses. Suatu proses, misalnya menentukan apakah suatu password yang dimasukkan benar atau salah, dapat diimplementasikan seperti berikut¹ (pseudocode):

Algorithm 4 Password verification.

Input: $\tilde{P} = (\tilde{P}[0], \dots, \tilde{P}[7])$ (and $P = (P[0], \dots, P[7])$)

Output: ‘true’ or ‘false’

```
1: for  $j = 0$  to  $7$  do
2:   if  $(\tilde{P}[j] \neq P[j])$  then return ‘false’
3: end for
4: return ‘true’
```

Pseudocode tersebut membandingkan setiap karakter dari password yang sebenarnya (P) dengan password yang dimasukkan (\tilde{P}), dan jika tidak sama langsung return. “Informasi sampingan” yang dapat diperoleh misalnya jumlah iterasi (looping) yang dilakukan dan juga waktu. Misalnya password yang benar adalah “p4\$\$w0rd”, dan password yang dimasukkan adalah “password”. Huruf kedua dari password yang dimasukkan tidak sama dengan password yang benar, sehingga program return setelah 2 iterasi. Dengan menghitung jumlah iterasi ini, penyerang bisa mencoba-coba huruf dari password dan mengecek apakah jumlah iterasinya meningkat, yang menandakan huruf tersebut benar. Jika penyerang tidak memiliki akses untuk menghitung jumlah iterasi (misalnya targetnya remote), penyerang bisa mengukur waktu yang diperlukan program untuk memberikan respon (*timing attack*), karena jumlah iterasi yang meningkat juga akan sedikit meningkatkan waktu eksekusi.

Challenge CTF kategori reverse engineering terkadang memiliki pola seperti ini, yaitu adanya fungsi pengecekan input yang langsung return (atau exit) begitu ditemukan ketidaksesuaian. Jika demikian, maka dapat dimanfaatkan side-channel attack untuk mendapatkan input yang benar, bahkan tanpa perlu menganalisis algoritma dari program. Jadi, analisis dilakukan secara “black box”.

Salah satu tools yang dapat digunakan untuk keperluan tersebut adalah **Pin** yang dikeluarkan oleh Intel². Pin merupakan tools untuk instrumentasi binary, dalam artian mengukur atau mencatat beberapa aspek dari binary secara dinamis ketika runtime (dengan

¹ <http://cs.ucsb.edu/~koc/cren/docs/w05/ch13.pdf>

² <https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool>

menjalankan binary-nya, tanpa perlu mengubah source code). Pin memiliki banyak “modul” yang dapat digunakan untuk keperluan instrumentasi. Untuk kasus reverse engineering pengecekan password seperti yang telah disebutkan, dapat digunakan modul “*instruction count*” (inscount³). Inscount akan menghitung jumlah instruksi (assembly) yang dijalankan program jika diberikan input tertentu. Dengan menghitung perbedaan jumlah instruksi jika diberikan input yang berbeda, dapat diketahui input yang benar.

Instalasi Intel Pin

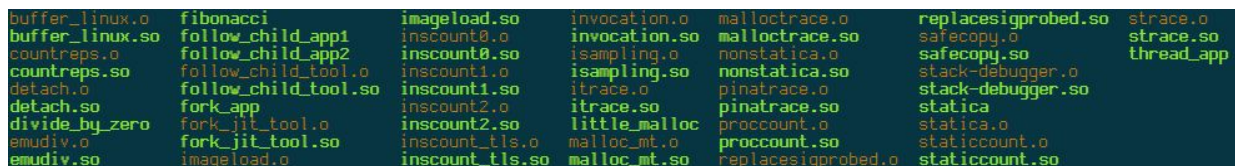
Download source code pintools di web Intel⁴ (33 MB). Lalu ekstrak file .gz:

```
$ tar xzf pin-3.0-76991-gcc-linux.tar.gz
```

Masuk ke folder source lalu compile:

```
$ cd pin-3.0-76991-gcc-linux/source/tools/ManualExamples
$ make
```

Jika sudah di-compile maka akan ada folder **obj-intel64** yang berisi hasil compile source code-nya. Di situ terdapat banyak tools (atau sebut saja modul) yang sudah dapat digunakan untuk dynamic analysis. Untuk tulisan ini digunakan modul `inscount1.so` untuk menghitung jumlah instruksi.



Studi Kasus CTF: Counter (Ekoparty 2015)

Challenge ini meminta password yang benar sebagai flag. Pertama download binary:

```
$ curl -s http://termbin.com/iocy | base64 -d > counter
```

Lalu jalankan programnya:

```
$ chmod +x counter
$ ./counter
```

³ <https://software.intel.com/sites/landingpage/pintool/docs/49306/Pin/html/index.html#SimpleCount>

⁴ <https://software.intel.com/en-us/articles/pintool-downloads>

```
usage: ./counter password
```

Password dimasukkan melalui command-line argument. Karena tutorial ini berfokus pada analisis binary secara dinamis maka kita tidak perlu menganalisis hasil disassemble decompile dari programnya, tapi cukup menjalankannya saja di bawah Pin.

Hitung instruksi ketika program dijalankan dengan memasukkan password "A":

```
$ env -i ./pin -t source/tools/ManualExamples/obj-intel64/inscount1.so --  
./counter A
```

Fungsi command "env -i" yaitu untuk ignore segala environment variable agar tidak mempengaruhi jalannya program inscount. Output terdapat pada file "inscount.out".

```
$ cat inscount.out  
  
Count 1019394
```

Jika password diisi "B":

```
$ env -i ./pin -t source/tools/ManualExamples/obj-intel64/inscount1.so --  
./counter B; cat inscount.out  
  
Count 1019394
```

Perhatikan jumlah instruksinya sama. Setelah dicoba-coba ternyata instruksi selalu sama kecuali huruf "S", sehingga dapat disimpulkan huruf "S" adalah karakter pertama dari password.

```
$ env -i ./pin -t source/tools/ManualExamples/obj-intel64/inscount1.so --  
./counter S; cat inscount.out  
  
Count 1612831
```

Proses bisa diotomatisasi dengan membuat program (dalam contoh ini Python) untuk melakukan bruteforce per karakter untuk menemukan password-nya. Seluruh kemungkinan huruf yang mungkin untuk password ada pada variabel `charset`.

bruteCounter.py

```
#!/usr/bin/python  
  
import string  
import commands  
charset = "  
_{}" + string.ascii_letters + string.digits
```

```
def main():
    password = ""
    for char in charset:
        tmp = password + char
        commands.getstatusoutput("env -i ./pin -t
source/tools/ManualExamples/obj-intel64/inscount1.so -- ./counter "+tmp)
        fd = open("inscount.out", "r")
        count = int(fd.read().split(" ")[1])
        fd.close()
        print '(+) count : ', count, tmp

if __name__ == "__main__":
    main()
```

Jika dijalankan:

```
$ python bruteCounter.py

(skip...)
(+) count : 1019394 K
(+) count : 1019394 L
(+) count : 1019394 M
(+) count : 1019394 N
(+) count : 1019394 O
(+) count : 1019394 P
(+) count : 1019394 Q
(+) count : 1019394 R
(+) count : 1612831 S
(+) count : 1019394 T
(+) count : 1019394 U
(skip...)
```

Karakter pertama yang benar adalah: "S"

Lalu ubah variable password di program dengan password yang sudah diketahui sejauh ini:

```
(skip...)

def main():
    password = "S"

(skip...)
```

Jalankan kembali program bruteforce:

```
$ python bruteCounter.py

(skip...)
(+) count : 1612831 Sh
(+) count : 1612831 Si
```

```
(+) count : 1612831 Sj
(+) count : 1612831 Sk
(+) count : 1612831 Sl
(+) count : 1612831 Sm
(+) count : 1612831 Sn
(+) count : 1612831 So
(+) count : 1612831 Sp
(+) count : 1612831 Sq
(+) count : 1612831 Sr
(+) count : 1612831 Ss
(+) count : 2167181 St
(+) count : 1612831 Su
(+) count : 1612831 Sv
(+) count : 1612831 Sw
(+) count : 1612831 Sx

(skip...)
```

Didapatkan kelanjutan password: "St". Lakukan terus proses sampai ditemukan password yang valid. Proses bruteforce per karakter ini membutuhkan waktu yang cukup lama (sekitar 30 menit, termasuk mengubah script Python untuk menambahkan password yang sudah diketahui sejauh ini) karena terdapat 1 juta instruksi, ditambah lagi terdapat cost tambahan dari Pin untuk menghitung instruksi dari program.

Untuk lebih otomatisnya, dapat dibuat script untuk menebak karakter:

```
#!/usr/bin/python

import string
import commands
not_appear_char = "~" # character that shouldn't appear
charset = "_{}" + string.ascii_letters + string.digits

def main():
    password = ""
    while 1:
        tmp = password + not_appear_char
        commands.getstatusoutput("env -i ./pin -t
source/tools/ManualExamples/obj-intel64/inscount1.so -- ./counter "+tmp)
        fd = open("inscount.out", "r")
        count = int(fd.read().split(" ")[1])
        fd.close()
        start = count

        detect = 0
        for char in charset:
            tmp = password + char
            commands.getstatusoutput("env -i ./pin -t
source/tools/ManualExamples/obj-intel64/inscount1.so -- ./counter "+tmp)
            fd = open("inscount.out", "r")
            count = int(fd.read().split(" ")[1])
            fd.close()
            print '(+) count : ', count, tmp
            if count != start:
                password = tmp; detect = 1
```

```
        break
    print "password:", password
    if not(detect):
        break

if __name__ == "__main__":
    main()
```

Studi Kasus CTF: Wirth (Idsecconf Offline 2016)

Download binary:

```
$ curl -s http://termbin.com/mejm | base64 -d > wirth
```

Lalu jalankan programnya:

```
$ ./wirth
flag : TEST
incorrect flag

$ ./wirth
flag : HELLO
incorrect flag
```

Ternyata program wirth meminta input dari STDIN, kemudian coba hitung instruksinya.

```
$ env -i ./pin -t source/tools/ManualExamples/obj-intel64/inscount1.so --
./wirth; cat inscount.out
flag : a
incorrect flag
Count 182490

$ env -i ./pin -t source/tools/ManualExamples/obj-intel64/inscount1.so --
./wirth; cat inscount.out
flag : b
incorrect flag
Count 182490

$ env -i ./pin -t source/tools/ManualExamples/obj-intel64/inscount1.so --
./wirth; cat inscount.out
flag : f
correct flag
Count 182480
```

Karena kita tahu bahwa format flag adalah flag{...} maka dapat dipastikan string pertama adalah **flag{**

```

$ env -i ./pin -t source/tools/ManualExamples/obj-intel64/inscount1.so --
./wirth; cat inscount.out
flag : fj
incorrect flag
Count 182517

$ env -i ./pin -t source/tools/ManualExamples/obj-intel64/inscount1.so --
./wirth; cat inscount.out
flag : fk
incorrect flag
Count 182517

$ env -i ./pin -t source/tools/ManualExamples/obj-intel64/inscount1.so --
./wirth; cat inscount.out
flag : fl
correct flag
Count 182507

```

Perhatikan selisih jumlah instruksi antara input karakter yang salah dan karakter yang benar, yang ternyata bernilai tetap (tidak selalu seperti ini):

```

>>> 182490-182480
10
>>> 182517-182507
10

```

Buat script untuk mencari offset kondisi benar dan kondisi salah:

findoffset.py
<pre> #!/usr/bin/python import string import commands charset = "_{}" + string.ascii_letters + string.digits def main(): for c in charset: # save guessed character to file fd = open("guess", "w") fd.write(c + '\n') fd.close() commands.getstatusoutput("env -i ./pin -t source/tools/ManualExamples/obj-intel64/inscount1.so -- ./wirth < guess") # read instruction count fd = open("inscount.out", "r") count = int(fd.read().split(" ")[1]) fd.close() print '[+] count : ', count, c if __name__ == "__main__": main() </pre>

Karena program meminta input dari STDIN maka kita akan buat file dahulu lalu di-input via STDIN ke programnya.

```
$ python findoffset.py
```

```
(skip...)  
[+] count : 182447 a  
[+] count : 182447 b  
[+] count : 182447 c  
[+] count : 182447 d  
[+] count : 182447 e  
[+] count : 182471 f  
[+] count : 182447 g  
[+] count : 182447 h  
[+] count : 182447 i  
[+] count : 182447 j  
(skip...)
```

Menghitung offset:

```
offset = true_condition - false_condition  
        = 182471 - 182447  
        = 24
```

Buat program otomatis:

bruteWirth.py

```
#!/usr/bin/python  
  
import string  
import commands  
charset = "_{}" + string.ascii_letters + string.digits  
offset = 24 # true condition - false condition  
tmp = ""  
cm = ""  
def main():  
    state = 0  
    flag = ""  
    while 1:  
        detect = 0  
        for c in charset:  
            tmp = flag + c  
            fd = open("guess", "w")  
            fd.write(tmp + '\n')  
            fd.close()  
            commands.getstatusoutput("env -i ./pin -t  
source/tools/ManualExamples/obj-intel64/inscount1.so -- ./wirth < guess")  
            fd = open("inscount.out", "r")  
            count = int(fd.read().split(" ")[1])  
            fd.close()  
            print '[+] count : ', count, c  
            if count - state == -offset:
```



```

        flag += cm; detect = 1
        break
    elif count - state == offset:
        flag += c; detect = 1
        break

    state = count
    cm = c

    print "flag:", flag
    if not(detect):
        break

if __name__ == "__main__":
    main()

```

Jalankan script, *and see it in action*: <https://asciinema.org/a/emzxhzuo11ta7hv934wn2pk8b>.
 Pada contoh ini proses bruteforce berjalan cukup cepat.

Studi Kasus CTF: Why (CompFest 2016)

Download binary:

```
$ curl -s http://termbin.com/moyp | base64 -d > why
```

Lalu jalankan:

```

$ ./why
[?] Enter password : HELLO
[ :) ] Congratulations! The password you input is nowhere near
right!

$ ./why
[?] Enter password : TEST
[ :) ] Congratulations! The password you input is nowhere near
right!

```

Mencari offset:

findoffset.py

```

#!/usr/bin/python

import string
import commands
charset = "_{}" + string.ascii_letters + string.digits

def main():
    for c in charset:
        # save guessed character to file
        fd = open("guess", "w")

```

```

fd.write(c + '\n')
fd.close()

commands.getstatusoutput("env -i ./pin -t
source/tools/ManualExamples/obj-intel64/inscount1.so -- ./why < guess")

# read instruction count
fd = open("inscount.out", "r")
count = int(fd.read().split(" ")[1])
fd.close()
print '[+] count : ', count, c

if __name__ == "__main__":
    main()

```

Jalankan script untuk mencari offset:

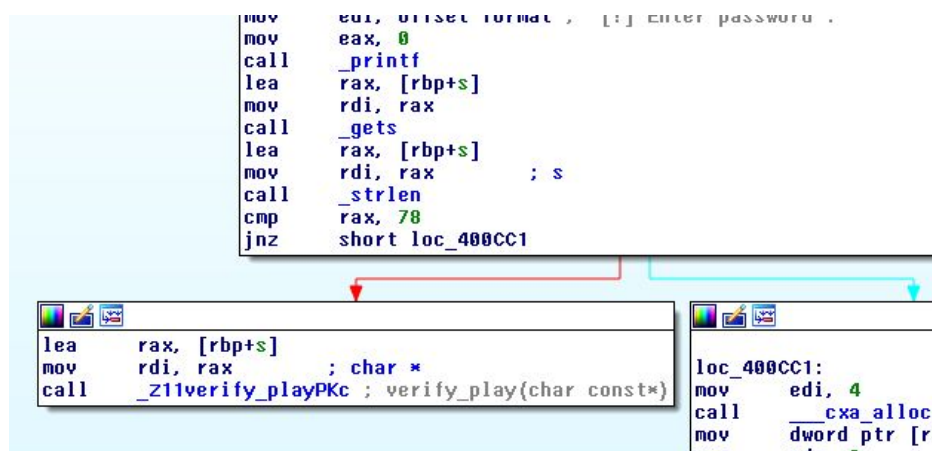
```

$ python findoffset.py

[+] count : 1422144 _
[+] count : 1422144 {
[+] count : 1422144 }
[+] count : 1422144 a
[+] count : 1422144 b
[+] count : 1422144 c
(skip..)
[+] count : 1422144 8
[+] count : 1422144 9

```

Namun, tidak ada karakter yang membuat jumlah instruksi berubah. Sepertinya harus sedikit ditambahkan pendekatan static analysis. Disassemble program dengan IDA dan perhatikan Flow Graph-nya.



Pada saat program berjalan, program akan memastikan input panjangnya 78 karakter (`call _strlen; cmp rax, 78`), jika iya maka akan dilakukan pengecekan password.

Teknik lainnya yaitu dengan pendekatan black box. Teknik ini melihat jumlah instruksi seiring pertambahan panjang karakter yang dimasukkan.

```
findlen.py

#!/usr/bin/python

import string
import commands

charset = "_{}" + string.ascii_letters + string.digits

def main():
    i = 1
    while i:
        tmp = "A"*i

        # save guessed password to file
        fd = open("guess", "w")
        fd.write(tmp + '\n')
        fd.close()

        commands.getstatusoutput("env -i ./pin -t
source/tools/ManualExamples/obj-intel64/inscount1.so -- ./why < guess")

        # read instruction count
        fd = open("inscount.out", "r")
        count = int(fd.read().split(" ")[1])
        fd.close()

        print '[+] count : ', count, i
        i+=1

if __name__ == "__main__":
    main()
```

Perhatikan pola ini:

```
$ python findlen.py

(skip...)
[+] count : 1422307 66
[+] count : 1422311 67
[+] count : 1422306 68
[+] count : 1422310 69
[+] count : 1422311 70
[+] count : 1422315 71
[+] count : 1422292 72
[+] count : 1422306 73
[+] count : 1422307 74
[+] count : 1422311 75
[+] count : 1422306 76
[+] count : 1422310 77
[+] count : 1565775 78
[+] count : 1422315 79
[+] count : 1422317 80
[+] count : 1422321 81
```

(skip...)

Ketika panjang input 78 karakter maka jumlah instruksi naik drastis, yang menandakan bahwa panjang 78 karakter kemungkinan adalah panjang input yang diinginkan oleh program.

Modifikasi script findoffset.py, memastikan bahwa panjang input harus 78 karakter:

```
findoffset.py

#!/usr/bin/python

import string
import commands

charset = "_{}" + string.ascii_letters + string.digits

def main():
    flag = "A"*78
    for c in charset:
        tmp = list(flag)
        tmp[0] = c
        tmp = "".join(tmp)

        # save guessed password to file
        fd = open("guess", "w")
        fd.write(tmp + '\n')
        fd.close()

        commands.getstatusoutput("env -i ./pin -t
source/tools/ManualExamples/obj-intel64/inscount1.so -- ./why < guess")

        # read instruction count
        fd = open("inscount.out", "r")
        count = int(fd.read().split(" ")[1])
        fd.close()

        print '[+] count : ', count, tmp

if __name__ == "__main__":
    main()
```

Jalankan script:

[illegible]

```
[+] count : 1565775
KAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[+] count : 1565775
LAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[+] count : 1565775
MAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[+] count : 1565775
NAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[+] count : 1565775
OAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[+] count : 1565775
PAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
(skip...)
```

Selisih offset = 1703781-1565775 = **138006**.

Buat program bruteforce otomatis untuk menebak 78 karakter yang benar:

bruteWhy.py

```
#!/usr/bin/python

import string
import commands

charset = string.ascii_letters + string.digits + "_{}"
offset = 137641
def main():

    flag = "A"*78
    res = {}
    for i in range(0, 79):
        n=0
        state = 0
        for c in charset:
            tmp = list(flag)
            tmp[i] = c

            fd = open("flag", "w")
            fd.write("".join(tmp)+'\n')
            fd.close()

            commands.getstatusoutput("env -i ./pin -t
source/tools/ManualExamples/obj-intel64/inscount1.so -- ./why < flag")

            fd = open("inscount.out", "r")
            count = int(fd.read().split(" ")[1])
            fd.close()
            res[c] = count

        if state == 0:
            state = 1
            n = count
        print "".join(tmp), str(count)
        if state == 1 and (count-n)>=offset:
            flag = "".join(tmp)
            print "flag: "+flag
            break
        elif state == 1 and abs(count-n)>=offset:
            print "flag: "+flag
            break
```

```
if __name__ == "__main__":  
    main()
```

Jalankan script, *and see it in action*: <https://asciinema.org/a/8o67cck6ituajv73xnm3k0epw>.

Pada kasus ini proses bruteforce memerlukan waktu lama, karena selain panjang input yang cukup panjang (78 karakter), juga jumlah instruksi yang besar (orde jutaan, dibandingkan kasus “Wirth” yang hanya ratusan ribu).

Penutup

Apakah teknik side-channel analysis ini selalu bisa digunakan? Jawabannya tentu tidak, karena supaya teknik ini bekerja, fungsi pengecekan input harus memenuhi kriteria seperti pada bagian pendahuluan. Teknik ini tidak akan bekerja jika fungsi pengecekan inputnya jumlah iterasinya konstan (berapapun panjang input, akan dibandingkan sepanjang password, dan tidak langsung return meskipun ketemu huruf yang salah). Pada kasus yang seperti itu, input salah atau benar jumlah instruksinya akan selalu sama (atau tidak dapat dicari perbedaan yang bermakna).